12 | 套路篇：CPU 性能优化的几个思路

2018-12-17 倪朋飞

12 | 套路篇：CPU 性能优化的几个思路
朗读人：冯永吉　14′46″ | 13.54M

你好，我是倪朋飞。

上一节我们一起回顾了常见的 CPU 性能指标，梳理了核心的 CPU 性能观测工具，最后还总结了快速分析 CPU 性能问题的思路。虽然 CPU 的性能指标很多，相应的性能分析工具也很多，但理解了各种指标的含义后，你就会发现它们其实都有一定的关联。

顺着这些关系往下理解，你就会发现，掌握这些常用的瓶颈分析套路，其实并不难。

在找到 CPU 的性能瓶颈后，下一步要做的就是优化了，也就是找出充分利用 CPU 的方法，以便完成更多的工作。

今天，我就来说说，优化 CPU 性能问题的思路和注意事项。

性能优化方法论

在我们历经千辛万苦，通过各种性能分析方法，终于找到引发性能问题的瓶颈后，是不是立刻就要开始优化了呢？别急，动手之前，你可以先看看下面这三个问题。

- 首先，既然要做性能优化，那要怎么判断它是不是有效呢？特别是优化后，到底能提升多少性能呢？
- 第二，性能问题通常不是独立的，如果有多个性能问题同时发生，你应该先优化哪一个呢？
- 第三，提升性能的方法并不是唯一的，当有多种方法可以选择时，你会选用哪一种呢？是不是总选那个最大程度提升性能的方法就行了呢？

如果你可以轻松回答这三个问题，那么二话不说就可以开始优化。

比如，在前面的不可中断进程案例中，通过性能分析，我们发现是因为一个进程的**直接 I/O**，导致了 iowait 高达 90%。那是不是用“**直接 I/O 换成缓存 I/O**”的方法，就可以立即优化了呢？

按照上面讲的，你可以先自己思考下那三点。如果不能确定，我们一起来看看。

- 第一个问题，直接 I/O 换成缓存 I/O，可以把 iowait 从 90% 降到接近 0，性能提升很明显。
- 第二个问题，我们没有发现其他性能问题，直接 I/O 是唯一的性能瓶颈，所以不用挑选优化对象。
- 第三个问题，缓存 I/O 是我们目前用到的最简单的优化方法，而且这样优化并不会影响应用的功能。

好的，这三个问题很容易就能回答，所以立即优化没有任何问题。

但是，很多现实情况，并不像我举的例子那么简单。性能评估可能有多重指标，性能问题可能会多个同时发生，而且，优化某一个指标的性能，可能又导致其他指标性能的下降。

那么，面对这种复杂的情况，我们该怎么办呢？

接下来，我们就来深入分析这三个问题。

怎么评估性能优化的效果？

首先，来看第一个问题，怎么评估性能优化的效果。

我们解决性能问题的目的，自然是想得到一个性能提升的效果。为了评估这个效果，我们需要对系统的性能指标进行量化，并且要分别测试出优化前、后的性能指标，用前后指标的变化来对比呈现效果。我把这个方法叫做性能评估“三步走”。

1. 确定性能的量化指标。
2. 测试优化前的性能指标。

3. 测试优化后的性能指标。

先看第一步，性能的量化指标有很多，比如 CPU 使用率、应用程序的吞吐量、客户端请求的延迟等，都可以评估性能。那我们应该选择什么指标来评估呢？

我的建议是**不要局限在单一维度的指标上**，你至少要从应用程序和系统资源这两个维度，分别选择不同的指标。比如，以 Web 应用为例：

- 应用程序的维度，我们可以用**吞吐量和请求延迟**来评估应用程序的性能。
- 系统资源的维度，我们可以用 **CPU 使用率**来评估系统的 CPU 使用情况。

之所以从这两个不同维度选择指标，主要是因为应用程序和系统资源这两者间相辅相成的关系。

- 好的应用程序是性能优化的最终目的和结果，系统优化总是为应用程序服务的。所以，必须要使用应用程序的指标，来评估性能优化的整体效果。
- 系统资源的使用情况是影响应用程序性能的根源。所以，需要用系统资源的指标，来观察和分析瓶颈的来源。

至于接下来的两个步骤，主要是为了对比优化前后的性能，更直观地呈现效果。如果你的第一步，是从两个不同维度选择了多个指标，那么在性能测试时，你就需要获得这些指标的具体数值。

还是以刚刚的 Web 应用为例，对应上面提到的几个指标，我们可以选择 ab 等工具，测试 Web 应用的并发请求数和响应延迟。而测试的同时，还可以用 vmstat、pidstat 等性能工具，观察系统和进程的 CPU 使用率。这样，我们就同时获得了应用程序和系统资源这两个维度的指标数值。

不过，在进行性能测试时，有两个特别重要的地方你需要注意下。

第一，要避免性能测试工具干扰应用程序的性能。通常，对 Web 应用来说，性能测试工具跟目标应用程序要在不同的机器上运行。

比如，在之前的 Nginx 案例中，我每次都会强调要用两台虚拟机，其中一台运行 Nginx 服务，而另一台运行模拟客户端的工具，就是为了避免这个影响。

第二，避免外部环境的变化影响性能指标的评估。这要求优化前、后的应用程序，都运行在相同配置的机器上，并且它们的外部依赖也要完全一致。

比如还是拿 Nginx 来说，就可以运行在同一台机器上，并用相同参数的客户端工具来进行性能测试。

多个性能问题同时存在，要怎么选择？

再来看第二个问题，开篇词里我们就说过，系统性能总是牵一发而动全身，所以性能问题通常也不是独立存在的。那当多个性能问题同时发生的时候，应该先去优化哪一个呢？

在性能测试的领域，流传很广的一个说法是“二八原则”，也就是说 80% 的问题都是由 20% 的代码导致的。只要找出这 20% 的位置，你就可以优化 80% 的性能。所以，我想表达的是，**并不是所有的性能问题都值得优化**。

我的建议是，动手优化之前先动脑，先把所有这些性能问题给分析一遍，找出最重要的、可以最大程度提升性能的问题，从它开始优化。这样的好处是，不仅性能提升的收益最大，而且很可能其他问题都不用优化，就已经满足了性能要求。

那关键就在于，怎么判断出哪个性能问题最重要。这其实还是我们性能分析要解决的核心问题，只不过这里要分析的对象，从原来的一个问题，变成了多个问题，思路其实还是一样的。

所以，你依然可以用我前面讲过的方法挨个分析，分别找出它们的瓶颈。分析完所有问题后，再按照因果等关系，排除掉有因果关联的性能问题。最后，再对剩下的性能问题进行优化。

如果剩下的问题还是好几个，你就得分别进行性能测试了。比较不同的优化效果后，选择能明显提升性能的那个问题进行修复。这个过程通常会花费较多的时间，这里，我推荐两个可以简化这个过程的方法。

第一，如果发现是系统资源达到了瓶颈，比如 CPU 使用率达到了 100%，那么首先优化的一定是系统资源使用问题。完成系统资源瓶颈的优化后，我们才要考虑其他问题。

第二，针对不同类型的指标，首先去优化那些由瓶颈导致的，性能指标变化幅度最大的问题。比如产生瓶颈后，用户 CPU 使用率升高了 10%，而系统 CPU 使用率却升高了 50%，这个时候就应该首先优化系统 CPU 的使用。

有多种优化方法时，要如何选择？

接着来看第三个问题，当多种方法都可用时，应该选择哪一种呢？是不是最大提升性能的方法，一定最好呢？

一般情况下，我们当然想选能最大提升性能的方法，这其实也是性能优化的目标。

但要注意，现实情况要考虑的因素却没那么简单。最直观来说，**性能优化并非没有成本**。性能优化通常会带来复杂度的提升，降低程序的可维护性，还可能在优化一个指标时，引发其他指标的异常。也就是说，很可能你优化了一个指标，另一个指标的性能却变差了。

一个很典型的例子是我将在网络部分讲到的 DPDK（Data Plane Development Kit）。DPDK 是一种优化网络处理速度的方法，它通过绕开内核网络协议栈的方法，提升网络的处理能力。

不过它有一个很典型的要求，就是要独占一个 CPU 以及一定数量的内存大页，并且总是以 100% 的 CPU 使用率运行。所以，如果你的 CPU 核数很少，就有点得不偿失了。

所以，在考虑选哪个性能优化方法时，你要综合多方面的因素。切记，不要想着“一步登天”，试图一次性解决所有问题；也不要只会“拿来主义”，把其他应用的优化方法原封不动拿来用，却不经过任何思考和分析。

CPU 优化

清楚了性能优化最基本的三个问题后，我们接下来从应用程序和系统的角度，分别来看看如何才能降低 CPU 使用率，提高 CPU 的并行处理能力。

应用程序优化

首先，从应用程序的角度来说，降低 CPU 使用率的最好方法当然是，排除所有不必要的工作，只保留最核心的逻辑。比如减少循环的层次、减少递归、减少动态内存分配等等。

除此之外，应用程序的性能优化也包括很多种方法，我在这里列出了最常见的几种，你可以记下来。

- **编译器优化**：很多编译器都会提供优化选项，适当开启它们，在编译阶段你就可以获得编译器的帮助，来提升性能。比如， gcc 就提供了优化选项 -O2，开启后会自动对应用程序的代码进行优化。
- **算法优化**：使用复杂度更低的算法，可以显著加快处理速度。比如，在数据比较大的情况下，可以用 O(nlogn) 的排序算法（如快排、归并排序等），代替 O(n^2) 的排序算法（如冒泡、插入排序等）。
- **异步处理**：使用异步处理，可以避免程序因为等待某个资源而一直阻塞，从而提升程序的并发处理能力。比如，把轮询替换为事件通知，就可以避免轮询耗费 CPU 的问题。
- **多线程代替多进程**：前面讲过，相对于进程的上下文切换，线程的上下文切换并不切换进程地址空间，因此可以降低上下文切换的成本。
- **善用缓存**：经常访问的数据或者计算过程中的步骤，可以放到内存中缓存起来，这样在下次用时就能直接从内存中获取，加快程序的处理速度。

系统优化

从系统的角度来说，优化 CPU 的运行，一方面要充分利用 CPU 缓存的本地性，加速缓存访问；另一方面，就是要控制进程的 CPU 使用情况，减少进程间的相互影响。

具体来说，系统层面的 CPU 优化方法也有不少，这里我同样列举了最常见的一些方法，方便你记忆和使用。

- **CPU 绑定**：把进程绑定到一个或者多个 CPU 上，可以提高 CPU 缓存的命中率，减少跨 CPU 调度带来的上下文切换问题。
- **CPU 独占**：跟 CPU 绑定类似，进一步将 CPU 分组，并通过 CPU 亲和性机制为其分配进程。这样，这些 CPU 就由指定的进程独占，换句话说，不允许其他进程再来使用这些 CPU。
- **优先级调整**：使用 nice 调整进程的优先级，正值调低优先级，负值调高优先级。优先级的数值含义前面我们提到过，忘了的话及时复习一下。在这里，适当降低非核心应用的优先级，增高核心应用的优先级，可以确保核心应用得到优先处理。
- **为进程设置资源限制**：使用 Linux cgroups 来设置进程的 CPU 使用上限，可以防止由于某个应用自身的问题，而耗尽系统资源。
- **NUMA（Non-Uniform Memory Access）优化**：支持 NUMA 的处理器会被划分为多个 node，每个 node 都有自己的本地内存空间。NUMA 优化，其实就是让 CPU 尽可能只访问本地内存。
- **中断负载均衡**：无论是软中断还是硬中断，它们的中断处理程序都可能会耗费大量的 CPU。开启 irqbalance 服务或者配置 smp_affinity，就可以把中断处理过程自动负载均衡到多个 CPU 上。

千万避免过早优化

掌握上面这些优化方法后，我估计，很多人即使没发现性能瓶颈，也会忍不住把各种各样的优化方法带到实际的开发中。

不过，我想你一定听说过高德纳的这句名言，“过早优化是万恶之源”，我也非常赞同这一点，过早优化不可取。

因为，一方面，优化会带来复杂性的提升，降低可维护性；另一方面，需求不是一成不变的。针对当前情况进行的优化，很可能并不适应快速变化的新需求。这样，在新需求出现时，这些复杂的优化，反而可能阻碍新功能的开发。

所以，性能优化最好是逐步完善，动态进行，不追求一步到位，而要首先保证能满足当前的性能要求。当发现性能不满足要求或者出现性能瓶颈时，再根据性能评估的结果，选择最重要的性能问题进行优化。

总结

今天，我带你梳理了常见的 CPU 性能优化思路和优化方法。发现性能问题后，不要急于动手优化，而要先找出最重要的、可以获得最大性能提升的问题，然后再从应用程序和系统两个方面入手优化。

这样不仅可以获得最大的性能提升，而且很可能不需要优化其他问题，就已经满足了性能要求。

但是记住，一定要忍住“把 CPU 性能优化到极致”的冲动，因为 CPU 并不是唯一的性能因素。在后续的文章中，我还会介绍更多的性能问题，比如内存、网络、I/O 甚至是架构设计的问题。

如果不做全方位的分析和测试，只是单纯地把某个指标提升到极致，并不一定能带来整体的收益。

思考

由于篇幅的限制，我在这里只列举了几个最常见的 CPU 性能优化方法。除了这些，还有很多其他应用程序，或者系统资源角度的性能优化方法。我想请你一起来聊聊，你还知道哪些其他优化方法呢？

欢迎在留言区跟我讨论，也欢迎你把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。

上一篇 11 | 套路篇：如何迅速分析出系统CPU的瓶颈在哪里？

写留言

精选留言

每天晒白牙 1

【D12打卡】
CPU性能优化思路
方法论
1.性能优化的效果判断
三步走理论
(1)确定性能的量化指标 - 一般从应用程序纬度和系统资源纬度分析
(2)测试优化前的性能指标
(3)测试性能优化后的性能指标
2.当性能问题有多个时，优先级问题
先优化最重要的且最大程度提升性能的问题开始优化
3.优化方法有多个时，该如何选
综合多方面因素
CPU优化
应用程序优化:排除不必要工作，只留核心逻辑
1.减少循环次数 减少递归 减少动态没错分配
2.编译器优化
3.算法优化
4.异步处理
5.多线程代替多进程
6.缓存
系统优化:利用CPU缓存本地性，加速缓存访问;控制进程的cpu使用情况，减少程序的处理速度
1.CPU绑定
2.CPU独占
3.优先级调整
4.为进程设置资源限制
5.NUMA优化
6.中断负载均衡
很重要的一点:切记过早优化

2018-12-17

C家族铁粉儿 1

又一篇精华满满的惊喜！
看到有人说，这些东西应该自己总结。是的，没错，很赞同。我也把用到的所有工具、指标、思路总结了一遍，但是看到老师给出了更全面系统的总结，仍然很受用。因为老师能从原理、关联多个角度给出更全面的知识网，也会指出一些易错的地方，是我们通学一遍、自己总结一遍以后的升华。起码，有些地方，因为原理知道的不多，我想错了或者漏了，看到这篇后，豁然开朗

老师给总结了，不等于你就会了，该学还得学，该记还得记，就像是满满的金子堆你面前，要不要弯腰去捡😊

2018-12-17

walker 0

想做一个全面的性能调优测试，不知道需要从哪些点入手，然后需要注意哪些问题？

2018-12-17

王涛 0

D12打卡。系统性优化是个大项目，要好好研究研究

2018-12-17

ninuxer 0

打卡，day13

cpu优化套路：

程序级别：编译器优化，算法优化，异步处理，多进程转多线程，善用缓存

系统级别：cpu绑定(优化上下文切换)，cpu独占（亲和性），nice值调整(得到更多的时钟周期)，cgroups设置资源限定，irqbalance负载中断，numa让cpu缓存命中率提升

2018-12-17

solar 0

打卡，打卡

2018-12-17

湖湘志 0

D12

2018-12-17